Experiment 1

Information Retrieval

Name:	**:**	**Abhishek N N**	**Register No**	**:**	**20BCE1025**
Faculty	:	Alok Chauhan	Slot	:	L51-L52 AB1-605B
Course	:	Web Mining Lab	Code	:	CSE3024
Programme	:	B.Tech CSE Core	Semester	:	Win – 22 - 23

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Exercise1.

- Explore Unix command GREP

grep command in Unix/Linux

The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for global search for regular expression and print out).

Syntax:

grep [options] pattern [files]

Options Description:

-c : This prints only a count of the lines that match a pattern

-h : Display the matched lines, but do not display the filenames.

-i : Ignores, case for matching

-l : Displays list of a filenames only.

-n : Display the matched lines and their line numbers.

-v : This prints out all the lines that do not matches the pattern

-e exp : Specifies expression with this option. Can use multiple times.

-f file : Takes patterns from file, one per line.

-E : Treats pattern as an extended regular expression (ERE)

-w : Match whole word

-o : Print only the matched parts of a matching line, with each such part on a separate output line.

-A n : Prints searched line and n lines after the result.

-B n : Prints searched line and n line before the result.

-C n : Prints searched line and n lines after before the result.

Consider the below file as an input.

```
20BCE1025 $ cat geekfile.txt
unix is great os. unix was developed in Bell labs.
learn operating system.
Unix linux which one you choose.
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
20BCE1025 $
```

1. **Case insensitive search:** The -i option enables to search for a string case insensitively in the given file. It matches the words like "UNIX", "Unix", "unix".

```
20BCE1025 $ grep -i "UNix" geekfile.txt
unix is great os. unix was developed in Bell labs.
Unix linux which one you choose.
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
20BCE1025 $
```

2. **Displaying the count of number of matches:** We can find the number of lines that matches the given string/pattern

```
20BCE1025 $ grep -c "unix" geekfile.txt
2
20BCE1025 $
```

3. **Display the file names that matches the pattern:** We can just display the files that contains the given string/pattern.

```
20BCE1025 $ grep -l "unix" *
geekfile.txt
20BCE1025 $
```

4. **Checking for the whole words in a file:** By default, grep matches the given string/pattern even if it is found as a substring in a file. The -w option to grep makes it match only the whole words.

```
20BCE1025 $ grep -w "unix" geekfile.txt
unix is great os. unix was developed in Bell labs.
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
20BCE1025 $
```

5. **Displaying only the matched patter :** By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.

```
20BCE1025 $ grep -o "unix" geekfile.txt
unix
unix
unix
unix
unix
20BCE1025 $
```

6. **Show line number while displaying the output using grep -n:** To show the line number of file with the line matched.

```
20BCE1025 $ grep -n "unix" geekfile.txt
1:unix is great os. unix was developed in Bell labs.
4:uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
20BCE1025 $
```

7. **Inverting the pattern match:** You can display the lines that are not matched with the specified search string pattern using the -v option.

```
20BCE1025 $ grep -v "unix" geekfile.txt
learn operating system.
Unix linux which one you choose.
20BCE1025 $
```

8. **Matching the lines that start with a string:** The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.

```
20BCE1025 $ grep "^unix" geekfile.txt
unix is great os. unix was developed in Bell labs.
20BCE1025 $
```

Exercise2.

i) Write a program to create the inverted index and execute for the following document collections.

a)

Doc 1 new home sales top forecasts
Doc 2 home sales rise in july
Doc 3 increase in home sales in july
Doc 4 july new home sales rise

b)

Doc 1 breakthrough drug for schizophrenia

Doc 2 new schizophrenia drug

Doc 3 new approach for treatment of schizophrenia

Doc 4 new hopes for schizophrenia patients

ii) Generate term-document incidence matrix for a) and b).

iii) For the document collections shown in b), compute the results for these queries using Boolean retrieval model:

- schizophrenia AND drug
- for AND NOT (drug OR approach)

Inverted Indexing Algorithm:

1) Map Data Structure is used for Indexing
   i. Initialize a map with key as string and value as set (doc.freq is set length) set is used because a word may repeat itself but doc id should not repeat
2) For each document in given directory give one doc id and repeat step 3) to 5)
3) Split the doc into strings and for For each string repeat step 4) to 5)
4) If string is not present in map assign doc id as a set
5) Else append doc id to existing set
6) Save dictionary as file for future

Time Complexity: O(n log(n))

Length of the total string is n and log(n) of time is required to insert 1 string

Space Complexity: O(n^2)

O(n) for term and doc.freq in map and for posting set O(n^2)

Term-document incidence matrix generation Algorithm:

1) Make mxn matrix m for terms and n for document and initialize with zero
2) For each key (term) and value (posting set) in Inverted Index map repeat 3) to
3) For each doc id in posting set repeat 4)
4) In matrix set 1 in row as term and column as respective posting set element
5) Print the matrix

Time Complexity: O(n)

Total number of entries in matrix is n, and we travel and update the matrix only once

Space Complexity: O(1)

Any additional space not used except input and output (matrix is representation of output)

Boolean retrieval Algorithm:

1) Get postfix query from infix query (by user)
2) Evaluate postfix query for each sub query repeat 3) to 4)
3) If term Get posting set by Inverted Indexing map (read from stored file)
4) Else if Boolean do set operation such as union intersection not
5) Display final output

Time Complexity: O(k*log(n))

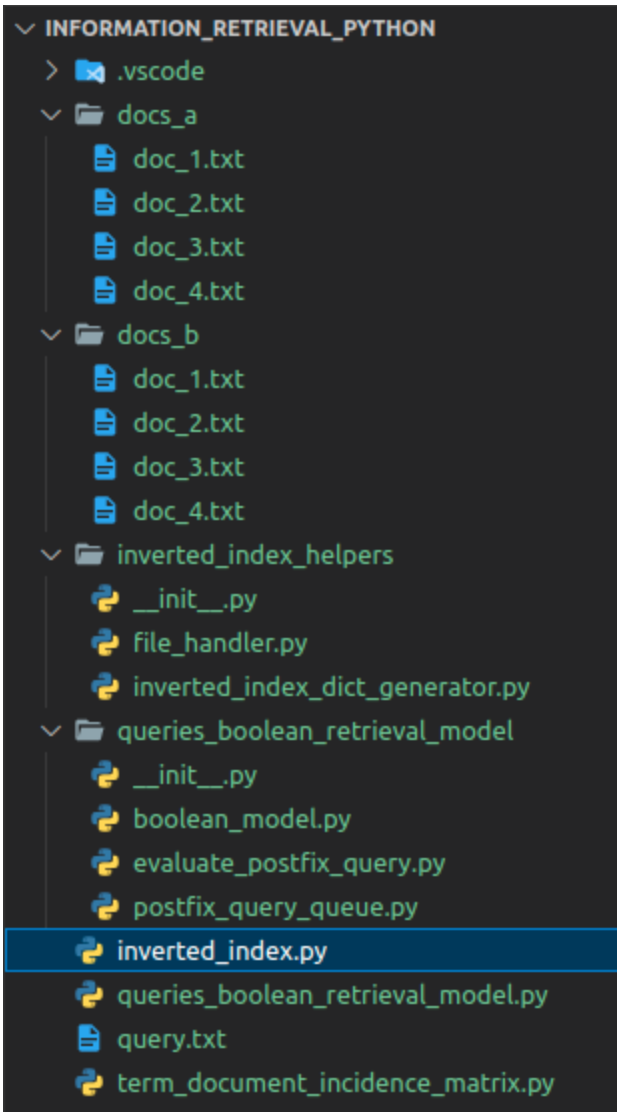
k is the total number of sub queries, and n is the no of terms in map

Space Complexity: O(n)

For generating postfix and evaluating it stack and queue is used

Running and Outputs:

File structure:



To Generate Inverted Index:

Give command as

python3 inverted_index.py –s <save_file_name> -d <dir_where_doc_colletions_stored>

```
● 20BCE1025 (main)$ python3 inverted_index.py -s save_a -d ./docs_a

          Term         DocId
0         new        {0, 3}
1        home  {0, 1, 2, 3}
2       sales  {0, 1, 2, 3}
3         top           {0}
4   forecasts           {0}
5        rise        {1, 3}
6          in        {1, 2}
7        july     {1, 2, 3}
8    increase           {2}

● 20BCE1025 (main)$ python3 inverted_index.py -s save_b -d ./docs_b

             Term         DocId
0   breakthrough           {0}
1           drug        {0, 1}
2            for     {0, 2, 3}
3  schizophrenia  {0, 1, 2, 3}
4            new     {1, 2, 3}
5       approach           {2}
6      treatment           {2}
7             of           {2}
8          hopes           {3}
9       patients           {3}

○ 20BCE1025 (main)$
```

To generate term-document incidence matrix:

Give command as

python3 term_document_incidence_matrix.py -s <saved_file_name>

```
20BCE1025 (main)$ python3 term_document_incidence_matrix.py -s save_a

            doc_1.txt  doc_2.txt  doc_3.txt  doc_4.txt
new                 1          0          0          1
home                1          1          1          1
sales               1          1          1          1
top                 1          0          0          0
forecasts           1          0          0          0
rise                0          1          0          1
in                  0          1          1          0
july                0          1          1          1
increase            0          0          1          0

20BCE1025 (main)$ python3 term_document_incidence_matrix.py -s save_b

               doc_1.txt  doc_2.txt  doc_3.txt  doc_4.txt
breakthrough           1          0          0          0
drug                   1          1          0          0
for                    1          0          1          1
schizophrenia          1          1          1          1
new                    0          1          1          1
approach               0          0          1          0
treatment              0          0          1          0
of                     0          0          1          0
hopes                  0          0          0          1
patients               0          0          0          1

20BCE1025 (main)$
```

To compute the results for queries using Boolean retrieval model:

Give command as

python3 queries_boolean_retrieval_model.py -s <saved_file_name> -q <query_file>

```
20BCE1025 (main)$ python3 queries_boolean_retrieval_model.py -s save_b -q ./query.txt
 schizophrenia AND drug : ['doc_1.txt', 'doc_2.txt']

 for AND NOT ( drug OR approach ) : ['doc_4.txt']
20BCE1025 (main)$
```

Codes:

inverted_index_helpers **file_handler.py**

```
"""
file module for handling all file operations
"""
from os import listdir
from os.path import isfile, join


def getDataFromDocs(dir):
    """
    gets strings from docs

    parameters:

    dir (str) : directroy which contains all files

    return:
    list of str read from docs in the directory given by user

    """
    return [open(join(dir, f)).read() for f in
sorted(listdir(dir)) if isfile(join(dir, f))]

def getDocIDToDocNameMap(dir):
    """
    gets the map of docID to docName

    parameters:

    dir (str) : directroy which contains all files

    return:
    dict of docID to docName of docs in the directory given by
user

    """
    return {i:x for i, x in enumerate([f for f in
sorted(listdir(dir)) if isfile(join(dir, f))])}
```

inverted_index_helpers **inverted_index_dict_generator.py**

```
def generateInvertedIndexDict(dataFromDoc: list[str]) :
    """
    generates the inverted index dict

    parameters:
    dataFromDoc (list[str]) : list of strings read from docs

    return:
    dict of term to set of docIDs
    """

    d=dict()

    termsListFromDoc = [s.split() for s in dataFromDoc]

    for docId, termList in enumerate(termsListFromDoc):
        for term in termList:
            if term not in d:
                d[term]={docId}
            else:
                d[term].add(docId)

    return d
```

inverted_index.py

```
"""
main module for inverted_index which handles everything
"""
from inverted_index_helpers import file_handler,
inverted_index_dict_generator
import sys, pickle
import pandas as pd

dir, saveFileName = str, str

# get the directory and save file name from command line arguments
for i, arg in zip(*[iter(sys.argv[1:])]*2):
    if (i=="-d"):
        dir = arg
    elif (i=="-s"):
        saveFileName = arg


dataFromDoc = file_handler.getDataFromDocs(dir)
docIdToDocName = file_handler.getDocIDToDocNameMap(dir)

invertedIndexDict =
inverted_index_dict_generator.generateInvertedIndexDict(dataFromDo
c)

print("\n",pd.DataFrame(invertedIndexDict.items(),
columns=["Term", "DocId"]),"\n")

# save the inverted index dictionary and docIDToDocName to a file
with open(saveFileName, 'wb') as handle:
    pickle.dump([invertedIndexDict, docIdToDocName] , handle,
protocol=pickle.HIGHEST_PROTOCOL)
```

term_document_incidence_matrix.py

```
"""
main module for term_document_incidence_matrix which handles
everything
"""
import sys, pickle
import pandas as pd

saveFileName = str

# get the directory and save file name from command line arguments
for i, arg in zip(*[iter(sys.argv[1:])]*2):
    if (i=="-s"):
        saveFileName = arg

# read the inverted index dictionary and docIDToDocName from a
file
with open(arg, 'rb') as handle:
    invertedIndexDict, docIdToDocName = pickle.load(handle)

# create a term-document incidence matrix with pandas
df=pd.DataFrame(index=invertedIndexDict.keys(),columns=range(len(d
ocIdToDocName)),data=0)

# fill the matrix with 1s
for term in invertedIndexDict.keys():
    for docId in invertedIndexDict[term]:
        df.loc[term,docId]=1

df.columns=docIdToDocName.values()

print("\n",df,"\n")
```

queries_boolean_retrieval_model boolean_model.py

```
class BooleanModel():
    """
    This class implements the boolean model for information
retrieval.

    Attributes:
        docIds (set): set of all document ids

    Methods:
        andOperation(left_operand, right_operand): returns the
intersection of two sets
        orOperation(left_operand, right_operand): returns the
union of two sets
        notOperation(operand): returns the difference of two sets
    """
    def __init__(self,docIdToDocName):
        self.docIds = set(docIdToDocName.keys())

    @staticmethod
    def andOperation(left_operand, right_operand):
        return left_operand.intersection(right_operand)

    @staticmethod
    def orOperation(left_operand, right_operand):
        return left_operand.union(right_operand)

    @staticmethod
    def notOperation(self, operand):
        return self.docIds.difference(operand)
```

queries_boolean_retrieval_model **postfix_query_queue.py**

```
from queue import Queue


def getPostfixQueryQueue(tokenizedQuery):
    """
    Returns a queue that will be used to store the postfix query

    Parameters:
        tokenizedQuery (list): list of tokens in the query

    Returns:
        Queue: queue that will be used to store the postfix query
                queue ements are tuples of the form (token,
isOperator)
    """
    q = Queue()

    operators = {"(", ")", "OR", "AND", "NOT"}
    priority = {"OR": 0, "AND": 1, "NOT": 2}

    stack = []

    for token in tokenizedQuery:
        if token not in operators:
            q.put((token, 0))
        elif token == "(":
            stack.append(token)
        elif token == ")":
            while stack and stack[-1] != "(":
                q.put((stack.pop(), 1))
            stack.pop()
        else:
            while stack and stack[-1] != "(" and priority[token]
<= priority[stack[-1]]:
                q.put((stack.pop(), 1))
            stack.append(token)

    while stack:
        q.put((stack.pop(), 1))

    return q
```

queries_boolean_retrieval_model evaluate_postfix_query.py

```
from .boolean_model import BooleanModel

def evaluatePostfixQuery(queue, docIdToDocName,
invertedIndexDict):
    """Evaluate a postfix query on the given index.

    Args:
        query (list): A list of tokens representing a postfix
query.
        index (dict): The index to evaluate the query on.

    Returns:
        list: A list of document IDs matching the query.
    """
    bm = BooleanModel(docIdToDocName)
    stack = []
    while not queue.empty():
        token, isOperator = queue.get()
        if isOperator:
            if token == "NOT":
                operand = stack.pop()
                stack.append(bm.notOperation(bm,operand))
            else:
                right_operand = stack.pop()
                left_operand = stack.pop()
                if token == "AND":
                    stack.append(bm.andOperation(left_operand,
right_operand))
                elif token == "OR":
                    stack.append(bm.orOperation(left_operand,
right_operand))
        else:
            stack.append(invertedIndexDict[token])

    return stack.pop()
```

queries_boolean_retrieval_model.py

```
"""
main module for queries using Boolean retrieval model which
handles everything
"""
import sys, pickle

from queries_boolean_retrieval_model.postfix_query_queue import
getPostfixQueryQueue
from queries_boolean_retrieval_model.evaluate_postfix_query import
evaluatePostfixQuery

saveFileName, queryFileName = str, str

# get the directory and save file name from command line arguments
for i, arg in zip(*[iter(sys.argv[1:])]*2):
    if (i=="-s"):
        saveFileName = arg
    elif (i=="-q"):
        queryFileName = arg

# read the inverted index dictionary and docIDToDocName from a
file
with open(saveFileName, 'rb') as handle:
    invertedIndexDict, docIdToDocName = pickle.load(handle)

# read the querys from a file and storing in list
queryList = open(queryFileName).read().replace("(", "(
").replace(")", " )").split("\n")

for query in queryList:
    # split the query into tokens by space
    tokenizedQuery = query.split(" ")
    # get the postfix query queue
    q=getPostfixQueryQueue(tokenizedQuery)
    # evaluate the postfix query
    res=evaluatePostfixQuery(q, docIdToDocName, invertedIndexDict)
    # convert docIDs to docNames
    res=[docIdToDocName[docId] for docId in res]
    print("\n",query,":",res,"\n")
```